A Data Model for Automatically Generating Typographical Layouts

William F. Kraus
Adobe Incorporated, Seattle, Washington, USA
kraus@adobe.com

Abstract

This paper proposes a binary tree data model which can be used to facilitate the automatic generation of aesthetically pleasing text layouts in graphic designs. The primary focus of this work is on text phrases commonly used for titles and headers, where there is considerable artistic freedom in how text is arranged.

Typography as a Model for Design

The arrangement of text within a graphic design contributes significantly to both the aesthetics and communicative impact of that design. While much of the focus on text layout has been on algorithms for line-breaking that mirror traditional typesetting (Knuth and Plass 1981), there is a need for a more flexible representation of text layout that can both generate a wider variety of arrangements and be amenable to analysis and generation by machine learning techniques. This paper describes one such representation.

Graphic design is built upon three pillars: *semiotics*, *style*, and *arrangement*. *Semiotics* refers to the semantics conveyed by the elements in a design, which are intentionally chosen to invoke an emotion and convey a message. This symbolism - where a design element has a meaning - can be literal or figurative and is often culturally dependent. *Style* refers to the geometric shapes and colors that result in a distinctive physical appearance, independent of the content. Finally, *arrangement* refers to how elements are placed within the composition, dictating a design's overall visual flow.

All three components can influence the visual weight of individual elements. Semiotics affects visual weight by leveraging our reflexive cognitive reaction to certain stimuli, such as an erotic image. Style does the same by using contrasting colors and shapes that trigger our visual perception system. And finally, arrangement affects visual weight through relative size and positioning within the visual flow.

Text layout provides an excellent playground for investigating this space, as it presents a concise example of this three-pillar model. Specifically, the glyphs and words that make up the text are semiotic - they are shapes on the canvas that, taken together, symbolize concepts and ideas. Text has style through the application of font, size, and color. And finally, the individual glyphs and words comprising the text are arranged so they can be recognized and read - there is a clear visual flow.

A Representation for Typographical Layout

To automatically generate typographical layouts, a flexible data representation is needed, one able to manifest a wide variety of layouts while also being easily modifiable. The solution proposed in this paper is to represent typographical layouts as binary trees, where the leaf nodes represent the geometries of the individual text tokens within the layout.

Text as a Graphic Element

A typographical layout is built by executing a sequence of transformation operations (scaling, rotation, and alignment) that act on individual blocks of text. These blocks are geometric objects that have a width and a height, as well as font related properties such as ascent, descent, baseline, and kerning (Fig. 1). While the exact values for these properties depend on the underlying text and the font applied to that text, typographical layouts arrange geometries and not the text per se.

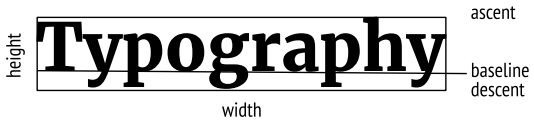


Figure 1. Each text token is represented as a geometric object.

Typographical Layout as a Binary Tree

Like any executable program, this sequence of transformation operations can be represented as a directed acyclic

graph (DAG) where each node specifies a text geometry, and each edge specifies an alignment operation between two nodes.

To simplify this representation so that it can be more easily modified without the risk of creating cycles, the DAG can be decomposed into a set of binary trees, one for each dimension. For two dimensional designs, a DAG would be decomposed into a horizontal and vertical binary tree, where each internal node is responsible for aligning its children in either the horizontal or vertical dimension.

To simplify this representation even further, these two trees can be merged into a single binary tree (Fig. 2) by using operators that specify alignments across both dimensions. For example, a left-to-right alignment operator in the horizontal binary tree and a top-to-bottom alignment operator in the vertical binary tree can be combined into a single left-to-right, top-to-bottom operator in the merged tree.

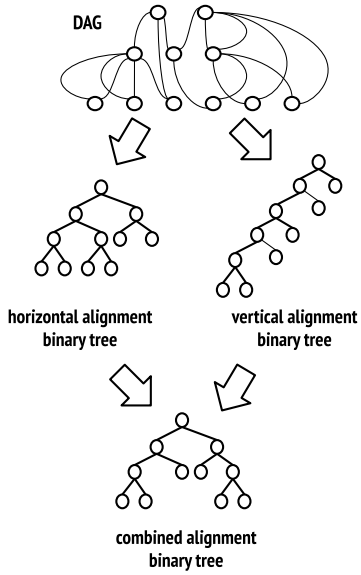


Figure 2. A typographical layout is represented as a single binary tree derived from a directed acyclic graph (DAG).

In addition, this single tree representation also simplifies support for scaling and rotation, as each node is associated with a transformation matrix (which would also include transposition due to alignment). Other properties such horizontal and vertical margins are assigned to a node.

An initial tree structure is built by first aligning adjacent words (or glyphs) into a run, where a run consists of a sequence of tokens all having the same weight (weight assignment is discussed below). Next, groups of runs are aligned, and then groups of groups, and so on until all text elements have been aligned. Algorithmically, a layout is generated by recursively walking down the tree starting at the root node, where each node first ensures both of its children are laid out before it attempts to align them to one another (Fig. 3).

While a given tree structure deterministically produces a given layout, it is possible that different tree structures can result in the same layout.

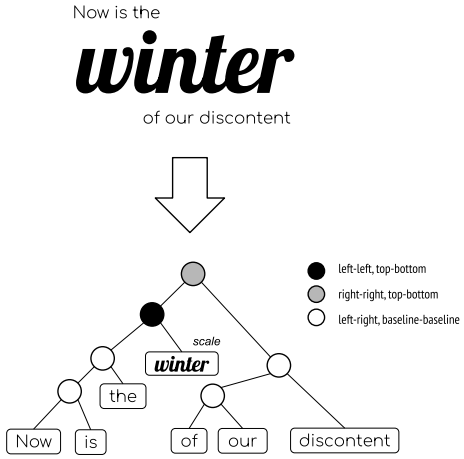


Figure 3. An example of a simple binary tree used to create one specific layout for the phrase 'Now is the winter of our discontent.' Three different 2D alignment operators are used here, one that aligns nodes top to bottom along their left edges, one that aligns nodes top to bottom along the right edges, and one that aligns nodes baseline to baseline and left to right. Note also that the node representing the word 'winter' is scaled.

While leaf nodes typically represent whole words, the binary tree representation is itself recursive, so that a word can itself be represented by a binary subtree where each leaf node is a separate glyph (and the alignment operators are language dependent). This partitioning of a word supports drop cap layouts, for example. Leaf nodes can also represent parts of words, and even non-textual content such as decorators or images or even empty space, which can be incorporated into the overall layout.

Styling and Weighting

While the binary tree representation arranges geometries, those geometries are dependent on the text and the font applied to that text. Changing the font or the range over which the font is applied requires a recalculation of the geometric values. However, doing so will not change the overall alignment specified by a given tree – it will simply result in the readjustment of the spacing between tokens and the size of tokens relative to one another.

A more interesting problem is how to determine which tokens should carry greater visual weight in a layout, either by increasing the relative size of a token, applying a differentiating style, or by placing the token in a prominent position within the visual flow of a layout. To support this concept, each leaf node in a binary tree is assigned a weight value, and each internal node inherits its weight from its children. Tree construction algorithms use this information

to determine which layouts are applicable, and which tokens should be prominent in those layouts.

Much as phonetic stress is used to emphasize certain words in speech, greater visual weight can be driven by semiotic considerations. By leveraging the results of natural language processing systems (NLP) such as spaCy (Honnibal, M., and Montani, I. 2017), leaf node weightings can be assigned based on NLP attributes such as parts-of-speech, relation within a dependency graph, and named entity recognition classifications. In addition, weights can be assigned based on correlations to other assets within the design - for example, a lemma for a word matches a tag associated with an image in the same design.

Automated Generation of Typographical Layouts

The binary trees that represent typographical layouts can be deterministically generated by implementing a 'layout program' for both building a tree and assigning properties to the individual nodes that make up that tree. Each program can be responsible for creating a single layout or multiple variations of a layout, depending on context.

An existing tree can also be used to generate new layouts by either modifying the properties associated with one or more nodes, and/or altering the tree structure itself (Figure 4). All the examples in Fig. 6 were programmatically generated this way.

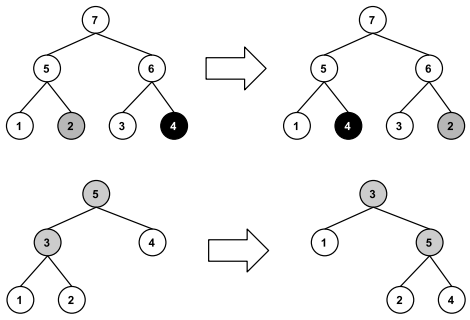


Figure 4. Novel layouts can be generated by modifying the tree structure. In the first case, leaf nodes are swapped. In the second case, the tree is rebalanced.

The binary tree representation also lends itself to various machine learning algorithms, since it provides an abstraction for describing typographical layouts that is only tangentially related to the text itself. This includes stochastic search algorithms, such as genetic programming (Koza 1990), where new layouts can be generated by exchanging subbranches between trees (Figure 5). This is facilitated by the constraint that in a binary tree representation, the number of 'group' nodes will always be *n - 1* where *n* is the number of text tokens in a phrase. This number will remain constant for a given text phrase regardless of how the text is arranged. In addition, in order to maintain the integrity of the original phrase, any exchange is constrained by the requirement that both subbranches must contain the same set of leaf nodes.

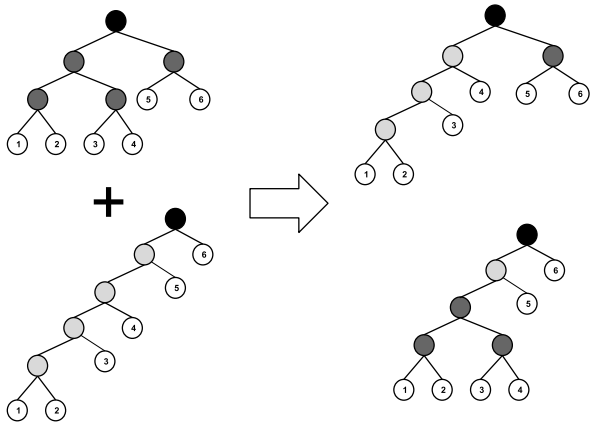


Figure 5. Layouts can exchange subbranches to generate novel layouts using stochastic search algorithms.

An initial investigation into using this technique on binary trees suggests that doing so can generate a large variety of unique and aesthetically pleasing layouts. However, it is also apparent from this preliminary work that the selection of an appropriate discriminator to evaluate and prioritize the layouts based on aesthetics, readability, and other design considerations is a necessity.

Summary

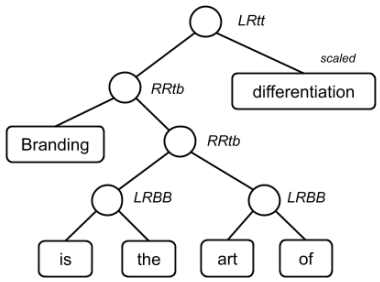
This paper describes a binary tree representation for typographical layout that lends itself to the automatic generation of a variety of graphically interesting designs and use by machine learning techniques.

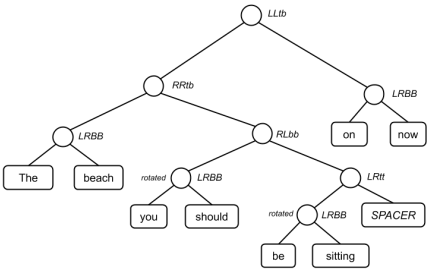
References

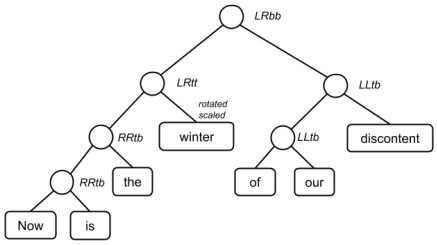
Honnibal, M., and Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing.

Knuth, Donald E., and Plass, Michael F. (1981), Breaking paragraphs into lines. *Software: Practice and Experience*, **11** (11): 1119–118

Koza, J. R. (1990). Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems. MIT Press, Cambridge, MA.







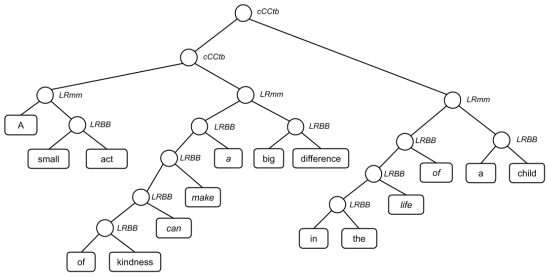


Figure 6. Examples of automatically generated layouts using a binary tree representation (shown below each graphic). Each node in the diagram is labeled with a mnemonic representing the alignment between its two children. For example, ‘LLtb’ is left-to-left, top-to-bottom whereas ‘LRBB’ is left-to-right, baseline-to-baseline.